Neural Network Architecture Optimization through Submodularity and Supermodularity

Junqi Jin, Ziang Yan, Kun Fu
Nan Jiang, Changshui Zhang
Dept. of Automation, Tsinghua University, Beijing, 100084, China
{jjq14, yza15, fuk11, jiangn15}@mails.tsinghua.edu.cn
zcs@mail.tsinghua.edu.cn

March 21, 2017

Abstract

Deep learning models' architectures, including depth and width, are key factors influencing models' performance, such as test accuracy and computation time. This paper solves two problems: given computation time budget, choose an architecture to maximize accuracy, and given accuracy requirement, choose an architecture to minimize computation time. We convert this architecture optimization into a subset selection problem. With accuracy's submodularity and computation time's supermodularity, we propose efficient greedy optimization algorithms. The experiments demonstrate our algorithm's ability to find more accurate models or faster models. By analyzing architecture evolution with growing time budget, we discuss relationships among accuracy, time and architecture, and give suggestions on neural network architecture design.

1 Introduction

Deep learning models have achieved breakthroughs in some machine learning tasks. Their hyper-parameters, including architecture, learning rate, weight decay, momentum etc., are important factors influencing the performance. In some cases, appropriate hyper-parameters are even more effective than novel approaches. However, most state-of-the-art models' hyper-parameters are hand-tuned for a particular data set. On one hand, reproducing the results is difficult, on the other hand, once data is changed, tuning again is compulsory. Thus, automatical and effective tuning hyper-parameter algorithms are important, which reveals a model's full potential while saving experts' tuning time.

We classify hyper-parameter optimization algorithms as follows. Random search does not utilize priors about hyper-parameters, nor takes advantage of evaluation history to decide next action. Compared to grid search, [1] theoretically and empirically tells that random search is more efficient. They reveal that only a few of hyper-parameters really matter, which means grid search wastes time evaluating unimportant dimensions while other dimensions unchanged. Random search is very practical because it's easy to implement and parallelize.

Bayesian Optimization (BO) and Tree-structured Parzen Estimator Approach (TPE) both model loss function respect to hyper-parameters in a probabilistic fashion. By maximizing expected improvement, these algorithms utilize evaluation history and balance exploration and exploitation well to determine next action. [28][29] show BO is very effective for general machine learning algorithms. [3][2] propose TPE method, different from Gaussian process modeling $p(y|x)$, which models $p(x|y), p(y)$ instead. (x is hyper-parameter and y is loss)

Compared to BO and TPE, [24] utilize more information. They compute exact gradients of the loss respect to hyper-parameters by chaining derivatives backwards through entire training procedure, which help optimize thousands of continuous hyper-parameters.

We focus on optimizing feed-forward neural network architecture which are discrete hyper-parameters. Most current hyper-parameter optimization algorithms are designed for general configurations tuning considered as black box functions, which do not utilize hyper-parameters' priors. The more information utilized, better results are obtained.

[24]'s gradient method takes much advantage of hyper-parameters' information, however, limited to continuous cases, which is not suitable for architecture choice problem. Thus, we propose methods that utilize both prior knowledge about hyper-parameters and evaluation history, which adopt marginal gain, a concept similar to discrete gradient to handle discrete hyper-parameters, such as hidden layers' widths.

A model's architecture, including each layer's width, is a key factor determining the model's performance. To maximize a model's accuracy, we have to pay cost, such as memory needed, power cost, etc. This paper chooses forward pass running time to measure a model's cost and solves two problems: given computation time budget, choose an architecture to maximize accuracy, and given accuracy requirement, choose an architecture to minimize computation time. Our contributions are as follows: (1) We propose an encoding method to convert this optimization into a subset selection problem. (2) We show that accuracy is submodular and running time is supermodular. (3) We propose two efficient greedy algorithms with theoretical bounds. (4) Surrogate functions are constructed using Gaussian process to speed up optimization. (5) Experiments demonstrate our methods achieve state-of-the-art by finding more accurate models or faster models. (6) By analyzing architecture evolution with growing time budget, we discuss relationships among accuracy, time and architecture, and give suggestions on neural network architecture design.

2 Approach

Our algorithm applies to general feed-forward neural networks. Given abundant practical models of convolutional neural network (CNN) achieving state-of-the-art performances in image and video recognition, natural language processing, we choose CNN as our main focus.

2.1 Convolutional Neural Network

Recently, CNN has become standard tools and achieves best results on MNIST [21], CIFAR [18], ImageNet [7], etc. CNN consists of many layers (depth), and each layer consists of many channels (widths). [19] won ILSVRC 2012 using a deep CNN. [33] designed GoogLeNet, which has 12 times fewer parameters than [19] while achieving better results, won ILSVRC 2014. [34] used Caffe reference network [12] yielding best results on Places scene Database. [27] proposed VGG net as one of ILSVRC 2014 winner. [23] suggested a micro network to take place of traditional convolution kernels.

Recent progress is derived from many factors, such as new math operation layer including inception module [33], network in network [23], or deeper model [27], or new large dataset [34].

Investigating architecture details of these state-of-the-art models, we find these CNNs' number of layers (depth) and each layer's neuron number (width) are mostly manually tuned. Most models' widths use numbers such as 96, 128, 256, 1024, 4096, which are manually set according to experience. [11] gave many suggestions about deep models' architecture, but mainly focused on non-linearity, pre-training. For a long time, the choice of depth and width is mostly by experience.

2.2 Architecture Optimization Problem

We consider two criteria, test accuracy and running time (cost needed to predict one image, not training time). There are many hyper-parameters affecting a CNN's accuracy and running time, such as convolution kernel size, learning rate, weight decay etc. This work mainly focuses on neural networks' depth and width. We denote CNN's architecture with k layers' depth as $\phi = [w_1, w_2, ..., w_k]$, where w_i is i_{th} layer's width (neuron number for full-connection layer, and channel number for convolution layer). Validation accuracy is denoted as A and running time as T. Once ϕ is determined, after training and validation, we get A and T values. Thus, A and T are functions of ϕ denoted as $A(\phi)$ and $T(\phi)$. ϕ forms CNN's architecture space $\Omega = \{[w_1, w_2...w_k] | k \geq 1, w_i \in \mathbb{Z}^+ \; for \; i = 1...k\}$. Our algorithms search Ω to solve two problems: given time budget B, maximize accuracy; given accuracy requirement R, minimize the running time, as Eq.1.

$$\max_{\phi} \; A(\phi) \quad s.t. \; T(\phi) \leq B \qquad\qquad \min_{\phi} \; T(\phi) \quad s.t. \; A(\phi) \geq R \tag{1}$$

Once depth is fixed, different widths' models are quantitatively related. However, different depths' models are qualitatively different. For example, $\phi_1 = [10, 20]$ and $\phi_2 = [10, 20, 5]$ are qualitatively different which cannot be optimized together, while $\phi_3 = [10, 20]$ and $\phi_4 = [10, 23]$ are strongly related. Therefore, we design the optimization as a two-step algorithm. First we traverse depths from 1 to D, for each depth we solve Eq.1 and return a best ϕ with that depth. Finally, the best ϕ across all depths is chosen. As the depth becomes large, the accuracy A converges, however, running time T keeps increasing. When D is large enough, we stop trying deeper models.

In a fixed depth case, Eq.1 is very hard to solve. For example, one of the best architectures LeNet-5 [20], includes 3 convolution stages and 1 full connection layer. The input and output layers have fixed widths, so we only consider above 4 tunable layers. Considering the scale of data, we set convolution channel with range $[1, 100]$, and full connection layer's width with range of $[1, 1000]$. Under the settings, traversing all possible ϕ means totally $100 \times 100 \times 100 \times 1000 = 1,000,000,000$ different architectures. Suppose evaluating a ϕ needs 1 second (actually several minutes or longer), $1,000,000,000$ costs 31 years! Let alone today's state-of-the-arts have much larger depth and width.

Since ϕ is a discrete vector, one of our contributions is to propose an architecture encoding method to convert this discrete optimization into a subset selection problem. LeNet-5 is $\phi = [6, 16, 120, 84]$, for convenience, we set convolution channel range as $[0, 127]$, and full connection layer range $[0, 1023]$. Then we are able to convert original decimal number into binary number, such as $w_1 = 0 \ to \ 127 \ \rightarrow \ '0000000' \ to \ '1111111'$. For $\phi = [6, 16, 120, 84]$, we concatenate each width's binary code as $'0000110\ 0010000\ 1111000\ 0001010100'$. There are totally 31 digits, in which we consider '0' or '1' digit as indicators of selection of a 31-element set $\mathbf{V}$. In the code, the i_{th} digit being '1' means the i_{th} element in $\mathbf{V}$ is selected, and digit '0' means not selected. Any group of selected elements forming subset $\mathbf{S} \subseteq \mathbf{V}$ corresponds to a 31-digit code, further corresponds to an architecture. Any architecture also corresponds to a subset $\mathbf{S}$. Therefore, the problems Eq.1 is re-formulated as a subset selection problem as Eq.2, where the cardinality of set $\mathbf{V}$ is the length of the binary code which is determined by the predefined value range of each layer's width.

$$\max_{\mathbf{S}\subseteq\mathbf{V}} A(\mathbf{S}) \quad s.t. \ T(\mathbf{S}) \leq B \qquad\qquad \min_{\mathbf{S}\subseteq\mathbf{V}} T(\mathbf{S}) \quad s.t. \ A(\mathbf{S}) \geq R \tag{2}$$

The encoding methods can be generalized. Suppose binary number c is encoded from some layer's w, from right to left, denote c's i_{th} digit as $c(i)$ (taking value 0 or 1). For a k-digit c, the conversion has a general form: $w = n_0 + \sum_{i=1}^{k} c(i) \cdot \delta(i)$. If any layer's w is too small, for example $w_2 = 1$, the whole architecture performs very bad. Therefore, we add an integer n_0 to ensure that even though all digits are 0, w starts from a reasonable number. A second advantage is that, once we know some layer's w has value larger than m (maybe from experience), setting $n_0 = m$ will speed up our algorithm without any loss of the final performance. We call the design of $\delta(i)$ weight policy. In previous case, $\delta(i) = 2^{i-1}$ is an exponential weight policy. If $\delta(i) = 1$ for all i, it's a linear weight policy. Linear policy results in the longest code, and meanwhile means finer search of Ω. Exponential policy gives the shortest code, meaning a coarse search in which layer widths jump by exponential values.

2.3 Submodularity and Supermodularity

Submodularity of $A(S)$ Borrowing techniques from [13], we check submodularity of $A(S)$. Through revising training procedure, we show $A(S)$ is monotone increasing. Consider $S_A \subset S_B$, our encoding method guarantees $w_{Ai} \leq w_{Bi}$ for all widths. Suppose S_A's CNN has parameter W_A which has shape $w_{A1} \times w_{A2}$ and S_B's corresponding W_B has shape $w_{B1} \times w_{B2}$. Because $w_{B1} \geq w_{A1}, w_{B2} \geq w_{A2}$, by setting W_B top-left sub-matrix as W_A and rest of W_B as zeros, S_B's CNN has exactly same computation procedure as S_A's CNN. Thus, we compare normally trained S_B and its subset CNN's accuracy, and retain better one as $A(S_B)$. Therefore, $A(S_A) \leq A(S_B)$ showing $A(S)$ is monotone increasing.

Using monotone spline interpolation, we construct $a(\cdot)$ for $A(\cdot)$, where $a(\cdot)$ satisfies: at discrete points a and A have same values; A is monotone increasing, and the property is maintained for a (first order derivatives $\forall i, a_{w_i} > 0$); a has second order derivatives. Fixing other layers' widths, only changing two layers' widths x, y, accuracy is denoted as $a(x, y)$, there's Thrm. 1. (proofs in supplementary) However, $a(x, y)$ may not satisfy non-positive second order derivatives. Thus, we transform Eq. 2's objective function as $\max_{S\subset V} \gamma(\mu A(S))$, where $\gamma(z) = 1 - exp(-z), \mu > 0$. Note that $\gamma(\cdot)$ is monotone increasing and $\mu > 0$, so transformed formulation has same optimum solution as Eq. 2. By requiring second order derivatives of $\gamma(\mu a(S))$ below or equal to zeros, we conclude

$\mu \geq \mu_0 = max_{x,y}(a_{xx}/a_x^2, a_{yy}/a_y^2, a_{xy}/a_x a_y)$ for all two-layer pairs, satisfying which, $\gamma(\mu A(S))$ is submodular according to Thrm. 1. For architectures (common CNN structure) and MNIST, CIFAR-10 data set in this paper, we estimate μ_0 using locally weighted quadratic regression (details see supplementary) and find μ_0 is always negative, which means original $A(S)$ is submodular, and transformation is not needed. We think the reason is that accuracy saturates very fast, even though when all widths increase, number of parameters increase quadratically. Submodularity means the set function $A(S)$: $2^V \rightarrow R$ for all subsets S_A, S_B, and $S_A \subseteq S_B \subseteq V$, $s \notin S_B, s \in V$, it holds that $A(S_A \cup s) - A(S_A) \geq A(S_B \cup s) - A(S_B)$.

Theorem 1. *If all two-layer pairs' second order derivatives satisfy $a_{xx} \leq 0, a_{yy} \leq 0, a_{xy} \leq 0$, original function $A(S)$ is submodular.*

Supermodularity of $T(S)$ $T(\phi)$ is evaluated in a serial fashion with CPU, thus, the total running time is the summation of every neuron's computation. Consider the k_{th} layer with w_k and $(k-1)_{th}$ layer with w_{k-1}. Each neuron in k_{th} layer accepts $(k-1)_{th}$ layer's w_{k-1} channels, and there are w_k neurons in k_{th} layer. Therefore, when accepting information from the previous layer, the k_{th} layer's computation is $C_1 \cdot w_{k-1} \cdot w_k$, where C_1 is a constant. Considering each layer has its own non-linear operation, in k_{th} layer, we add a term $C_2 \cdot w_k$. Thus, the total running time of a k-layer architecture $\phi = [w_1, w_2, ..., w_k]$ is $T(\phi) = [1, w_1, w_1 w_2, w_2, ..., w_{k-1}, w_{k-1} w_k, w_k]\beta$ where β is a positive vector. With this formulation, we prove that $T(S)$ is monotone increasing and supermodular. (proofs in supplementary) A supermodular function means that the set function $T(S)$: $2^V \rightarrow R$ for all subsets S_A, S_B, and $S_A \subseteq S_B \subseteq V$, $s \notin S_B, s \in V$, it holds that $T(S_A \cup s) - T(S_A) \leq T(S_B \cup s) - T(S_B)$.

2.4 Optimization Algorithms

Submodular Maximization Algorithms Submodular maximization algorithms [15][17][22] are very usefull in machine learning field, aiming to maximize a submodular function with different constraints [5][25][6][14][32][9][10].

Constrained Method We firstly consider $max_S\ A(S)\ s.t.\ T(S) < B$, in which different from above constraints, we maximize submodular $A(S)$ with supermodular constraint $T(S) < B$. Borrowing the ideas from [22], considering supermodular constraint, we propose Alg.1. Adapted from [14] [16], [13] proved Thrm.2 and Thrm.3 as bounds for Alg.1 (proofs in supplementary). $\kappa_T = \min_{j \in V} \frac{T(j)}{T(V) - T(V \backslash j)}$ is defined as curvature of supermodular function $T(S)$. Because $0 \leq \kappa_T \leq 1$, curvature measures the distance of $T(S)$ from modularity. And $\kappa_T = 1$, if and only if $T(S)$ is modular, or $T(S) = \sum_{j \in S} T(j)$.

Theorem 2. *If S^* is the solution from Algorithm 1 when budget is B. Then, $A(S^*) \geq \frac{1}{2}(1 - \frac{1}{e})A(S^{opt}_{small})$, where S^{opt}_{small} is the global optimum when budget is $\kappa_T B$.*

Thrm.2 gives an $A(S^*)$'s lower bound based on optimum of a smaller budget $\kappa_T B \leq B$ problem.

Theorem 3. *The solution S^* from Algorithm 1 when budget is B satisfy $A(S^*) \geq \frac{1}{m+1}(1 - \frac{1}{e})A(S^{opt})$, where S^{opt} is the global optimum when budget is B and $m = max\{|S| : T(S) \leq \frac{B}{\kappa_T}\}$.*

With a smaller approximation factor, Thrm.3 gives a lower bound based on the same budget B problem. κ_T decides value of m. And it proves that as κ_T becomes large, m becomes small, in modular case $\kappa_T = 1$, the bound becomes $A(S^*) \geq \frac{1}{2}(1 - \frac{1}{e})A(S^{opt})$.

Unconstrained Method $max_S\ A(S)\ s.t.\ T(S) < B$ is also solved by optimizing an unconstrained problem: $max\ A(S) - \lambda T(S)$, where $\lambda > 0$ measures trade-off between accuracy and running time. We have to tune λ to make sure $T(S) \leq B$ and increase the solution's $T(S)$ as large as possible. We introduce Thrm.4 5, based on which we propose methods to tune λ. (proofs in supplementary) Thrm. 4 suggests: if we find a λ and solve problem $max\ A(S) - \lambda T(S)$, and the solution S^* satisfies that $T(S^*) \leq B, |B - T(S^*)| \leq \epsilon$, where ϵ is very small, then with budget ϵ-tolerance, S^* is the optimum of problem $max\ A(S),\ s.t.\ T(S) \leq B$.

Initialize: $S_u \leftarrow \emptyset, U = V$
while $U \neq \emptyset$ **do**
  foreach $X \in U$, compute
    $\delta_X = A(S_u \cup X) - A(S_u)$
  $X^* = argmax\{\delta_X : X \in U\}$
  If $T(S_u \cup X^*) \leq B$, **then**
    $S_u = S_u \cup X^*$
  $U = U \setminus X^*$
end while
Initialize: $S_s \leftarrow \emptyset, U = V$
while $U \neq \emptyset$ **do**
  foreach $X \in U$, compute
    $\delta_X = \frac{A(S_s \cup X) - A(S_s)}{T(S_s \cup X) - T(S_s)}$
  $X^* = argmax\{\delta_X : X \in U\}$
  If $T(S_s \cup X^*) \leq B$, **then**
    $S_s = S_s \cup X^*$
  $U = U \setminus X^*$
end while
Return $S^* = argmax_{S \in \{S_u, S_s\}} A(S)$

Algorithm 1: Constrained Greedy

Initialize: $S_1 = \emptyset, S_2 = V$
for $i = 1$ to $|V|$ **do**
  $a = F(S_1 \cup X_i) - F(S_1)$
  $b = F(S_2 \setminus X_i) - F(S_2)$
  If $a \geq b$, **then** $S_1 = S_1 \cup X_i$, **else** $S_2 = S_2 \setminus X_i$
end for
Return S_1

Algorithm 2: Deterministic Unconstrained Greedy

Initialize: $S_1 = \emptyset, S_2 = V$
for $i = 1$ to $|V|$ **do**
  $a = max\{F(S_1 \cup X_i) - F(S_1), 0\}$
  $b = max\{F(S_2 \setminus X_i) - F(S_2), 0\}$
  Sample Y from $Uniform(0, 1)$
  If $Y \leq \frac{a}{a+b}$, **then** $S_1 = S_1 \cup X_i$, **else** $S_2 = S_2 \setminus X_i$
end for
Return S_1 (Note: if $a = b = 0$, assume $\frac{a}{a+b} = 1$)

Algorithm 3: Randomized Unconstrained Greedy

Theorem 4. *S^* is the optimum of problem $max\ A(S) - \lambda T(S)$. Define $B = T(S^*)$, then S^* is the optimum of problem $max\ A(S),\ s.t.\ T(S) \leq B$.*

Theorem 5. *Suppose $\lambda_1 < \lambda_2$ and S_1 is optimum of problem $max\ A(S) - \lambda_1 T(S)$ and S_2 is optimum of problem $max\ A(S) - \lambda_2 T(S)$, then $T(S_1) = B_1 \geq B_2 = T(S_2)$.*

Thrm.5 shows that, as λ increases, the problem $max\ A(S) - \lambda T(S)$'s optimum solution's $B = T(S)$ decreases. We firstly finds λ_1 corresponding $S_1^*, T(S_1^*) \leq B$ and λ_2 corresponding $S_2^*, T(S_2^*) \geq B$. Then binary search helps to find a feasible solution S_1^* which is near budget B. The unconstrained problem is still NP-hard. Thus, we use an approximate algorithm. Despite approximation, the $\lambda\ vs\ T(S^*)$'s relationship is approximately monotone. (A typical result see supplementary)

$A(S)$ is submodular, $T(S)$ is supermodular, so $F(S) = A(S) - \lambda T(S)$ is submodular when $\lambda > 0$. Note $A(S) - \lambda T(S)$ is non-monotone. According to [5], there are Alg.2 3 to maximize non-monotone submodular function $F(S)$. [5] gives Thrm.6 as bounds.

Theorem 6. *Suppose S^{opt} is the optimum to maximize $F(S)$. Then, solution S_1 from Algorithm 2 satisfies $F(S_1) \geq \frac{F(S^{opt})}{3}$. And solution S_1 from Algorithm 3 satisfies $E[F(S_1)] \geq \frac{F(S^{opt})}{2}$.*

Relationship of Two Problems We convert $\min_S T(S), s.t.\ A(S) \geq R$ as $\min_Z T(V \setminus Z), s.t.\ A(V \setminus Z) \geq R$ with substitution trick. This problem is same as $\max_Z T(V) - T(V \setminus Z), s.t.\ A(V) - A(V \setminus Z) \leq A(V) - R$. It proves that $P(Z) = T(V) - T(V \setminus Z)$ is monotone non-decrease submodular and $Q(Z) = A(V) - A(V \setminus Z)$ is monotone non-decrease supermodular. Denote $B_R = A(V) - R$, then two problems in Eq.2 are equivalent.

2.5 Bayesian Optimization

In Alg.1 2 3, we need to frequently query $A(S), T(S)$. Every query means training and validating an architecture, which is expensive. To speed up the procedure, based on evaluated points, we construct surrogate functions $\hat{A}(S)$ for $A(S)$ and $\hat{T}(S)$ for $T(S)$. Alg. 1 2 3 are conducted on $\hat{A}(S), \hat{T}(S)$. We alternately optimize over surrogate functions and evaluate new points based on information got from previous optimization to make surrogate functions more and more accurate in important regions. We adopt Bayesian optimization [4][28] to model this process.

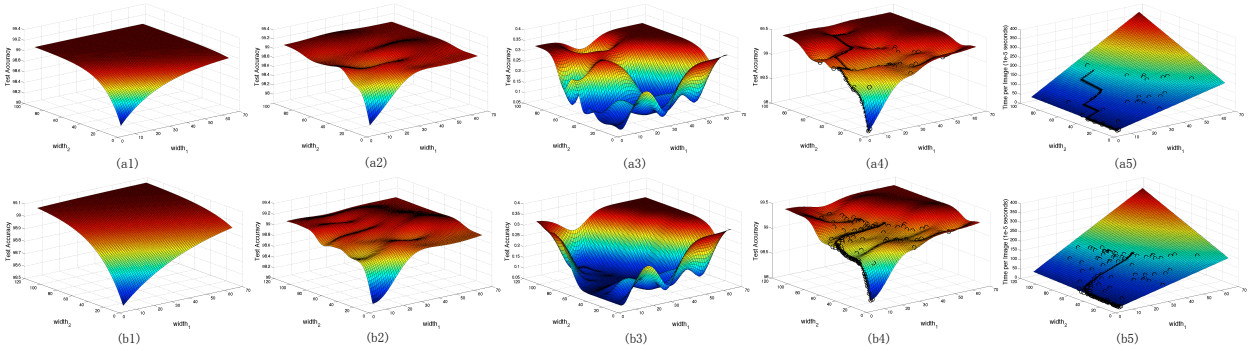

Figure 1: Optimizing a two-layer architecture. (a1)-(a5): 30 evaluations, (b1)-(b5): 80 evaluations. For both rows, (1): $A_p(S)$; (2): $A_p(S) + \mu_A(S)$; (3): $\sigma_A(S)$; (4): $\hat{A}(S)$; (5): $\hat{T}(S)$. (4)(5): greedy sequence from Algorithm 1 with marker X, circles are evaluated samples. Many evaluations near X-path make the path stable.

Gaussian Process with Priors According to [30], Gaussian process performs bad when modeling non-stationary functions. Considering submodularity and supermodularity, we design priors $A_p(S)$ for $A(S)$ and $T_p(S)$ for $T(S)$. If $\phi = [w_1, w_2, ..., w_k]$, the priors are as Eq.3, 4, where $C_1 > 0, C_2 > 0, C_3$ are scalars, $\beta_A \in R^k$ and $\beta_T \in R^{2k}$ are positive column vectors. We use least square loss $[A(S) - A_p(S)]^2, [T(S) - T_p(S)]^2$ and gradient descent to learn $C_1, C_2, C_3, \beta_A, \beta_T$.

$$A_p(S) = C_1 - C_2 \cdot exp(C_3 - [w_1, w_2...w_k] \cdot \beta_A) \tag{3}$$

$$T_p(S) = [1, w_1, w_1 \cdot w_2, w_2...w_{k-1} \cdot w_k, w_k] \cdot \beta_T \tag{4}$$

$A_p(S), T_p(S)$ predict $A(S), T(S)$'s trends well. Thus, residues $A(S) - A_p(S), T(S) - T_p(S)$ are stationary enough. We use Gaussian process to model residues. Suppose there are samples $\{\phi_n = [w_{n1}, w_{n2}...w_{nk}]\}_{n=1}^N$ with $\{A(\phi_n)\}_{n=1}^N$ and $\{T(\phi_n)\}_{n=1}^N$, denoting accuracy residues as $r_n = A(\phi_n) - A_p(\phi_n), n = 1...N$, it induces a multivariate Gaussian distribution on R^N: $[r_1, r_2...r_N]^T \sim \mathcal{N}(0, K)$, where $K_{ij} = \theta_0 \cdot exp(-\frac{\sum_{d=1}^k (w_{id} - w_{jd})^2 / \theta_d^2}{2}) + \mathbf{1}(i = j) \cdot \sigma^2, 1 \leq i, j \leq N$, in which $\mathbf{1}(\cdot)$ is indicator function. $T(\phi)$'s residues are similar. $\theta_0, \theta_d, \sigma$ are learnt by maximizing marginal likelihood of the multivariate Gaussian distribution which is commonly used [26][28]. Any samples' residues r, conditioned on other samples, satisfy a Gaussian distribution $\mathcal{N}(\mu, \Sigma)$, where μ and Σ are calculated using K and conditioned samples' residues. We denote a residue conditioned on collected samples as $A(S) - A_p(S) \sim \mathcal{N}(\mu_A(S), \sigma_A(S))$ and $T(S) - T_p(S) \sim \mathcal{N}(\mu_T(S), \sigma_T(S))$. Thus, the Gaussian process with priors gives Eq. 5.

$$A(S) \sim \mathcal{N}(A_p(S) + \mu_A(S), \sigma_A(S)) \qquad T(S) \sim \mathcal{N}(T_p(S) + \mu_T(S), \sigma_T(S)) \tag{5}$$

Upper-Confidence Bound (UCB) To balance exploitation and exploration, Eq. 5 concludes UCB as surrogate functions as Eq. 6, where β is chosen as settings in [31].

$$\hat{A}(S) = A_p(S) + \mu_A(S) + \beta^{\frac{1}{2}} \sigma_A(S) \qquad \hat{T}(S) = T_p(S) + \mu_T(S) - \beta^{\frac{1}{2}} \sigma_T(S) \tag{6}$$

Note that, in $\hat{A}(S)$ ($\hat{T}(S)$ is similar), $A_p(S)$ is monotone and submodular, however, $\mu_A(S) + \beta^{\frac{1}{2}} \sigma_A(S)$ relies on evaluated points which make $\hat{A}(S)$ not strictly submodular locally which overrates $A(S)$. However, once a point is evaluated, its variance is reduced and corresponding $\hat{A}(S)$ drops near true value. Since Alg. 1 2 3 are greedy methods, final results rely on optimization procedure. So we not only evaluate final optimum, but also intermediate points used in greedy selection which are randomly selected to evaluate. Bayesian optimization can automatically correct and improve the greedy selection. Consider in some step, the algorithm greedily choose a wrong element by overrating it, and by evaluating the intermediate points including this wrong element, its value drops in next optimization term and may be taken place by other elements. Wrong elements are always evaluated until some element, the best one, is discovered which has higher value than other elements no matter overrated or not. Then the greedy selection sequence becomes stable.

Fig. 1 illustrates this procedure. In Alg. 1, set S repeatedly climbs $\hat{A}(S)$ with $\hat{T}(S)$'s constraint. Final point and climbing path are evaluated to make used surface more accurate. UCB guides to explore valuable regions. With enough

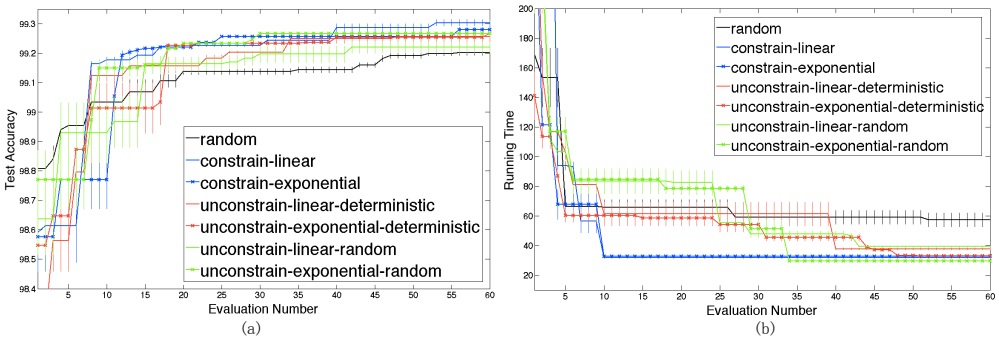

Figure 2: (a) $max_S A(S)$ with budget 50e-5 seconds; (b) $min_S T(S)$ with requirement 99.20.

evaluated points, the solution and path are stable, which are better than evaluated points deterministically and better than unevaluated points with high probability.

3 Experiment

Datasets and Other Hyper-parameters We use MNIST and CIFAR-10 datasets for classification tasks. MNIST is split into training, validation, and test sets with ratio 5:1:1, and for CIFAR-10 is 4:1:1. For each point's evaluation, grid search is used for learning rate in $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ and convolution kernel size in $\{3, 4, 5\}$. Pooling size is set 2. We use Caffe's SGD [12] with momentum 0.9. Weight matrices are initialized with zero-mean normal distribution whose variances are according to [8], and biases are initialized with 0. Training mini-batch size is 100. Early stopping is used. Validation running time is based on CPU processor clock.

3.1 Methods Comparison

We compare our different methods with random search as baseline [1]. Using MNIST, we optimize a two convolution stages architecture. Problem Eq.1 with budget 50e-5 seconds' results are as Fig.2(a), constrained method with linear policy achieves 99.30 higher than random search's 99.21. Problem Eq.1 with requirement 99.20's results are as Fig.2(b), unconstrained random method with exponential policy achieves 29.59 faster than random search's 57.46. All our methods perform better than baseline, and constrained methods (like climbing a hill) seem to be better than unconstrained methods (like encountering of climbing and going down a hill). Random search can achieve better results, however, needs more evaluations. Investigating the improvement of accuracy and running time, we find improving accuracy is more difficult than reducing running time.

3.2 Architecture Analysis

We optimize architectures with growing running time budget. In Fig.3(a), for MNIST and CIFAR-10, CNN's widths are optimized under each running time budget and best accuracies are plot in the figure. We tried different depths architectures, each of which corresponds to a curve. Any point under a curve means a worse architecture (same running time, lower accuracy). Each depth's architecture's accuracy saturates as model grows large, and deeper models have higher saturated accuracy. For each depth, we define the turning point architecture as minimum architecture of that depth, which has a saturated accuracy and lowest running time. We draw these minimum architectures in Fig.3(b). It tells that accuracy increases as the model goes deeper, however, once the model is deep enough, increasing accuracy a little needs running time growing very fast. This tells that improving accuracy is more difficult than reducing time. This phenomenon helps applications to estimate the accuracy and running time trade-off.

Having analyzed the results, we suggest an architecture design method. Firstly, try depths from shallow to deep. In each depth, set large widths to obtain saturated accuracy. If the accuracy meets the requirements, further to find the minimum architecture of this depth. When setting widths, the ratios of widths are important. We plot best architectures' widths evolution examples in Fig.3(c). We find different layers' widths are positively correlated; widths' ratios have relationship to data set, which cannot be concluded for all cases; we find lower layers (first and second layers) cost

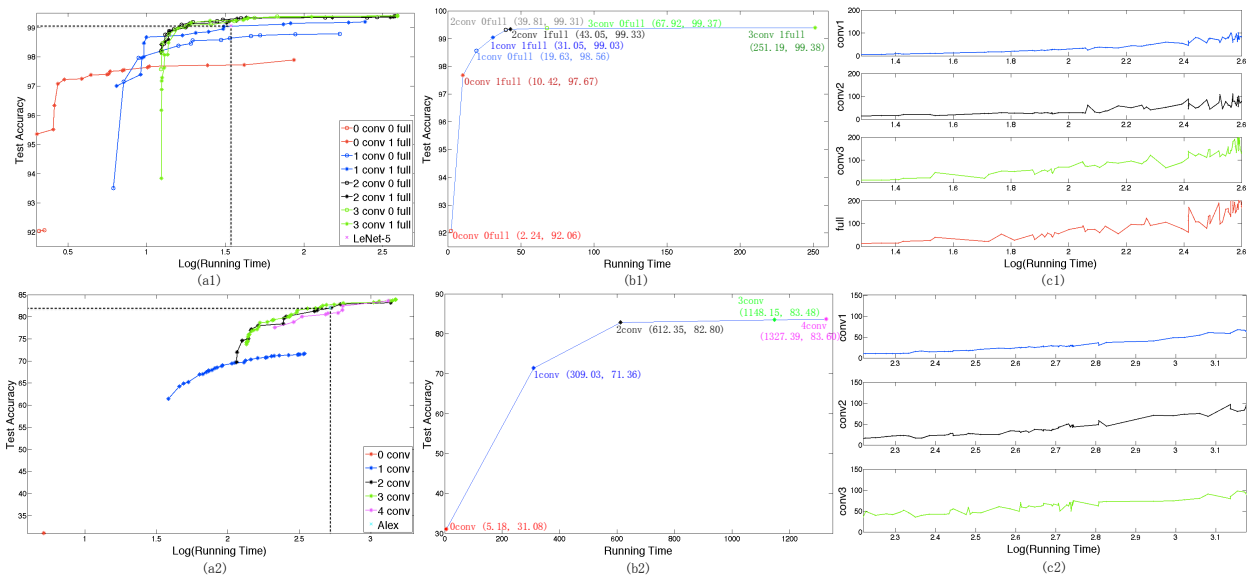

Figure 3: Row 1: MNIST; Row 2: CIFAR-10; (a): Different depths' best architectures' curves (LeNet-5, Alex's coordinates with dashed lines); (b): Minimum architectures; (c): Each layer's width evolution of best architectures (MNIST: 3 conv 1 full, CIFAR-10: 3 conv) (0conv0full: fed to softmax directly)

much running time, and our algorithms prefer lower layers with smaller widths (first width and second width ratio varies between 1:1.5 to 1:6); full connection layer costs less running time, usually has larger width, however, no more than 3 times of the last convolution width. Using above experience, we tune Alex Net [19] which won ILSVRC 2012. We only tune the first three widths using only center crop patch and get $\phi = [62, 210, 450]$ with top-1 accuracy 56.92, running time 16636e-5 seconds faster than Caffe's reproduced results [12] with $\phi = [96, 256, 384]$, top-1 accuracy 57.10, running time 21982e-5 seconds.

3.3 State of the Art

For state-of-the-art [20] LeNet-5, [18] Alex model, our algorithms find higher accuracy models with running time no increasing and faster models with accuracy not decreasing as Tab.1 2 ($width = 0$ means no this layer, time unit: 1e-5 seconds). The state-of-the-art and ours do not use data augmentation. Purely for high accuracy, in MNIST with budget 400, we get $\phi = [72, 88, 140, 0]$, running time 396.58 and accuracy 99.40, which is better than Boosted LeNet-4 with data augmentation. In CIFAR-10 with budget 1500, we get $\phi = [66, 82, 95]$, running time 1483.03 and accuracy 83.88, which is near results with data augmentation. [28] achieves 85.0 accuracy, however, they optimized learning rate, momentum, weight decay, etc. using Alex model, which does not conflict and can be combined with our architecture optimization.

Model	test acc	time	architecture
LeNet5	99.05	34.44	6, 16, 120, 84
higher1	99.26	30.53	10, 16, 0 , 0
higher2	99.21	31.34	6, 30, 48, 48
higher3	99.21	34.01	12, 14, 0 , 22
faster1	99.08	19.20	4, 25, 19, 22
faster2	99.06	22.96	4, 56, 0 , 0
faster3	99.08	18.63	4, 24, 0 , 68

Table 1: MNIST:higher accuracy and faster ϕ

Model	test acc	time	architecture
Alex	82.00	521.00	32, 32, 64
higher1	82.76	520.34	27, 46, 74
higher2	82.80	517.84	28, 43, 74
higher3	82.72	520.51	27, 47, 71
faster1	82.06	445.04	25, 37, 59
faster2	82.48	463.39	26, 36, 62
faster3	82.01	467.77	28, 33, 65

Table 2: CIFAR-10:higher accuracy and faster ϕ

4 Discussion

Our algorithms optimized neural network's architectures (widths and depths). Different from general methods, we introduced two priors, submodularity for accuracy and supermodularity for running time. Based on these properties, the optimization could be solved efficiently using greedy algorithms. We gave theoretical bounds for these algorithms, which are extensions of modular constraint cases. Our algorithms achieved better accuracy or running time than baseline, and proposed more accurate models or faster models than state-of-the-art. We proposed methods to design architectures by analyzing the evolution of best architectures as running time grew. We found accuracy saturated very fast as the model went deeper.

Our work mainly tunes the widths of layers, while other hyper-parameters using grid search or recommended settings. We found optimization of learning rate, momentum, etc. were helpful to achieve good results and could be combined with architectures optimization to get better performance. In future work, we will research on more hyper-parameters properties and utilize to guide designs of good architectures.

Acknowledgement

This work is supported by 973 Program(2013CB329503)

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, 2012.

[2] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *ICML*, pages 115–123, 2013.

[3] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, pages 2546–2554, 2011.

[4] Ali Borji and Laurent Itti. Bayesian optimization explains human active search. In *NIPS*, 2013.

[5] Niv Buchbinder, Michael Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *FOCS*, pages 649–658. IEEE, 2012.

[6] Niv Buchbinder, Moran Feldman, Joseph Seffi Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *ACM-SIAM*, pages 1433–1452. SIAM, 2014.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*. IEEE, 2009.

[8] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[9] Rishabh Iyer and Jeff Bilmes. Algorithms for approximate minimization of the difference between submodular functions, with applications. *Uncertainity in Artificial Intelligence*, 2012.

[10] Rishabh K Iyer and Jeff A Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In *NIPS*, pages 2436–2444, 2013.

[11] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*. IEEE, 2009.

[12] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[13] Junqi Jin, Ziang Yan, Kun Fu, Nan Jiang, and Changshui Zhang. Optimizing recurrent neural networks architectures under time constraints. *arXiv preprint arXiv:1608.07892*, 2016.

[14] Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[15] Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*, 3:19, 2012.

[16] Andreas Krause and Carlos Guestrin. A note on the budgeted maximization of submodular functions.

[17] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *JMLR*, 9:235–284, 2008.

[18] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.

[22] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *SIGKDD*, pages 420–429. ACM, 2007.

[23] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *ICLR*, 2014.

[24] Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Gradient-based hyperparameter optimization through reversible learning. *ICML*, 2015.

[25] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions-i. *Mathematical Programming*, 14(1):265–294, 1978.

[26] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

[28] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, pages 2951–2959, 2012.

[29] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md Patwary, Mostofa Ali, Ryan P Adams, et al. Scalable bayesian optimization using deep neural networks. *arXiv preprint arXiv:1502.05700*, 2015.

[30] Jasper Snoek, Kevin Swersky, Richard S Zemel, and Ryan P Adams. Input warping for bayesian optimization of non-stationary functions. *ICML*, 2014.

[31] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *ICML*, 2010.

[32] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[34] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NIPS*, pages 487–495, 2014.